Cruciverba numerico
Elaborato per l'esame di Intelligenza artificiale (B003725)

Studente: Valentina Proietti --- Aprile 2020

Abstract

Ci si propone di creare un modello per la soluzione di un cruciverba numerico in cui le soluzioni sono il prodotto dei valori della caselle, dato lo schema e le definizioni. Il modello è stato creato per l'ambiente Minizinc.

Definizione del problema

La soluzione del cruciverba numerico in questione consiste nel completare le caselle bianche con numeri interi positivi compresi tra due valori assegnati positivi e non nulli (r e s con r<s) in modo che il prodotto di celle bianche adiacenti dia una delle possibili definizioni, ogni definizione va usata una sola volta.

Il problema in Minizinc

In Minizinc la modellazione consiste in:

1) definizione dei parametri e delle variabili;
2) descrizione dei vincoli;
3) definizione dell'obiettivo;
4) eventuale descrizione di come dovranno essere scritti a video i risultati.

Anche se in Minizinc l'ordine in cui tutto questo viene scritto è ininfluente, per facilità di lettura ho preferito scriverlo in quest'ordine.

Definizione dei parametri e delle variabili (1)

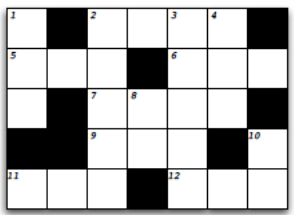
I parametri nel nostro caso sono r e s, lo schema del cruciverba e le definizioni.
Le variabili sono il contenuto delle caselle bianche.

Ho strutturato i dati nel seguente modo:

- r,s vengono semplicemente definiti;
- sono definiti in modo esplicito n e m (numero di righe e colonne del cruciverba);
- per inserire lo schema si utlizza un array nxm dove 1 indica casella nera e 0 indica casella da compilare, ad es.:

```
schema=[| 0,1,0,0,0,0,1
        | 0,0,0,1,0,0,0
        | 0,1,0,0,0,0,1
        | 1,1,0,0,0,1,0
        | 0,0,0,1,0,0,0
        |];
```

corrisponde a



- ho utilizzato una matrice noX4 (con no numero di soluzioni orizzontali) per avere indicazione di dove si trovano le definizioni nello schema e della loro lunghezza, per lo schema visto sopra si avrà ad esempio:

```
orizzontali=[| 2,1,3,4      // il 2 orizzontale inizia nella casella (1,3) ed e' lungo 4 caselle
             | 5,2,1,3      // il 5 orizzontale inizia nella casella (2,1) ed e' lungo 3
             | 6,2,5,3      //ecc.
             | 7,3,3,4
             | 9,4,3,3
             | 11,5,1,3
             | 12,5,5,3
            |];
```

  dove, per ogni riga, il primo valore indica il numero della definizione, il secondo e terzo indicano la posizione della casella di partenza della definizione e l'ultimo indica la sua lunghezza in caselle;
- allo stesso modo ho utilizzato una matrice nvX4 per le definizioni verticali;
- i valori no e nv vengono definiti esplicitamente;
- l'insieme delle definizioni possibili è definito come un set di interi.

Descrizione dei vincoli (2)

Ho predisposto degli assert per controllare che i dati in ingresso rispettino le condizioni indicate nel problema e che siano congruenti tra loro. In caso negativo apparirà a video l'indicazione di dove è il problema.

Per vincolare i valori delle celle ad assumere valori tra r e s ho scritto:

```
constraint forall (i in 1..n, j in 1..m)
                (crux[i,j]!=0 -> ((crux[i,j]<=s) /\ (crux[i,j]>=r)));
```

questo deve valere solo per le celle bianche (che per scelta costruttiva sono per noi quelle diverse da zero).

Ho deciso di utilizzare un array, che ho chiamato *solution*, composto da valori in S e di lunghezza pari a quella di S. Tale array viene vincolato ad essere semplicemente una permutazione delle possibili soluzioni riordinata secondo l'ordine di utilizzo (prima tutte le orizzontali poi tutte le verticali).

Ho quindi inserito il vincolo per le definizioni utilizzando l'array *solution*: per ogni riga in *orizzontali* la produttoria del valore contenuto nell'ultima casella della definizione, per quello della casella subito alla sua sinistra, per quello della casella ancora a sinistra fino ad esaurire la lunghezza della definizione (valore, come descritto precedentemente, indicato nella colonna 4 della matrice chiamata *orizzontali*).

```
constraint forall (i in 1..no)
 (
   product(

           [if (orizzontali[i,4]-j>0)
           then (crux[orizzontali[i,2],(orizzontali[i,3]+orizzontali[i,4]-j-1)])
           else 1 endif | j in 0..m]

    ) = solution[i]
 );
```

In modo analogo si procede per le definizioni verticali partendo dall'ultima casella della definizione e salendo.

L'ultimo vincolo da imporre è quello di utilizzare le definizioni una ed una sola volta. Includendo alldifferent.mzn si può scrivere semplicemente:

```
constraint alldifferent (solution);
```

che impone a tutti i componenti di alldifferent di essere diversi tra loro.

Definizione dell'obiettivo (3)

Il nostro obiettivo è quello di trovare una soluzione, quindi:

```
solve satisfy;
```

Formattazione risultati in uscita (4)

Ho preferito definire esplicitamente come visualizzare i risultati in modo da renderli più facilmente leggibili: ho fatto in modo che fosse visualizzato lo schema del cruciverba con le soluzioni inserite (gli zero sono caselle nere) e le definizioni riscritte nell'ordine corretto. Per far questo ho sfruttato il comando output, una condizione if/then/else per controllare quando andare a capo e varie concatenazioni di stringhe:

```
output [if (j=1) then "\n" else " " endif ++ show(crux[i,j]) |  i in 1..n,j in 1..m]++["\n\n"]++["ORIZZONTALI:"]++["\n"]++["\(orizzontali[i,1]).\(solution[i])\n"    |    i    in    1..no]++["\n"]++["VERTICALI:"]++["\n"]++["\(verticali[i-no,1]).\(solution[i])\n" | i in no+1..no+nv];
```

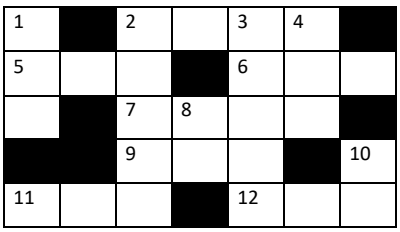
Test e Risultati

Per verificare il funzionamento del mio modello ho utilizzato 3 diversi cruciverba di prova: il primo è quello fornito con il testo del progetto, gli altri due li ho inventati a mano costruendo il cruciverba con i risultati inseriti in modo da avere un elenco di possibili definizioni (inventare le soluzioni a caso porta quasi sempre, inevitabilmente, a problemi insoddisfacibili) per poi reinserire come dati i loro schemi vuoti mantenendo il gruppo di definizioni.

<u>Data_numcross_vp.dnz</u>

```
S={6,8,24,28,30,36,42,54,64,70,84,90,96};r=1;s=9;
```

Restituisce come risultato:



8 0 1 2 3 7 0
6 5 3 0 1 5 6
2 0 4 8 1 2 0
0 0 7 1 4 0 6
9 6 1 0 3 8 1

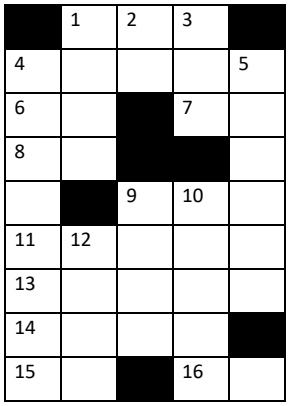
ORIZZONTALI:
2.42
5.90
6.30
7.64
9.28
11.54
12.24

VERTICALI:
1.96
2.84
3.36
4.70
8.8
10.6

Data2_numcross_vp.dzn

```
S={1,2,3,7,8,12,20,25,30,32,40,42,45,60,108,120,192,360,720}; r=1;s=9;
```

Restituisce come risultato:



0 1 5 6 0
6 1 9 2 1
1 7 0 5 5
2 6 0 0 4
1 0 4 8 6
1 4 5 1 6
5 8 1 1 1
3 1 1 1 0
2 1 0 1 1

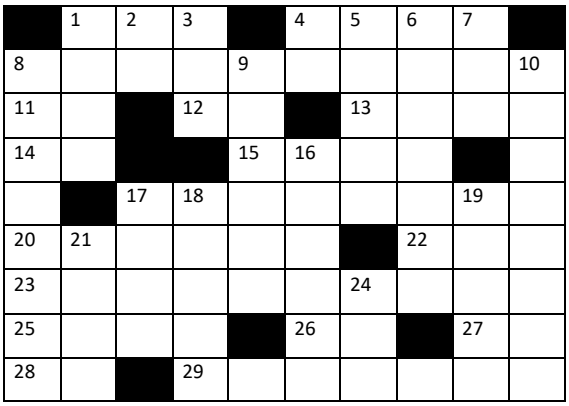
ORIZZONTALI:
1.30
4.108
6.7
7.25
8.12
9.192
11.120
13.40
14.3
15.2
16.1

VERTICALI:
1.42
2.45
3.60
4.360
5.720
9.20
10.8
12.32

Data3_numcross_vp.dzn

```
S={1,2,4,5,6,7,10,14,16,18,21,24,25,27,28,35,42,50,54,60,72,80,96,108,120,140,19
2,216,240,250,280,480,500}; r=1; s=9;
```

Restituisce come risultato:



0 1 7 6 0 9 2 2 6 0
1 1 3 2 1 2 5 1 2 1
4 7 0 9 3 0 5 1 6 8
5 5 0 0 4 5 5 5 0 6
1 0 7 5 1 1 1 1 4 1
1 7 2 2 5 2 0 8 3 4
4 1 1 1 1 1 4 1 1 1
6 1 1 1 0 5 1 0 2 1
1 1 0 1 6 1 1 9 1 1

ORIZZONTALI:
1.42
4.216
8.120
11.28
12.27
13.240
14.25
15.500
17.140
20.280
22.96
23.16
25.6
26.5
27.2
28.1
29.54

VERTICALI:
1.35
2.21
3.108
4.18
5.250
6.80
7.72
8.480
9.60
10.192
16.50
17.14
18.10
19.24
21.7
24.4